MetaComm: A Meta-Directory for Telecommunications

Juliana Freire[1] Daniel Lieuwen[1] Joann Ordille[1]
Lalit Garg[2] Michael Holder[2] Hector Urroz[2]
Gavin Michael[3] Julian Orbach[3] Luke Tucker[3] Qian Ye[1] Robert Arlein[1]

[1]*Bell Labs Research, 600 Mountain Ave., Murray Hill, NJ 07974*
{juliana,lieuwen,joann,qy,rma}@research.bell-labs.com
[2]*Bell Labs, 11900 N Pecos St, Westminster, CO 80234*
{lalit,mholder,hau}@lucent.com
[3]*Bell Labs, 15 Talavera Road, North Ryde, NSW 2113 Australia*
gavinm@au1.ibm.com, {juliano,luketucker}@lucent.com

Abstract

A great deal of corporate data is buried in network devices — such as PBX messaging/email platforms, and data networking equipment — where it is difficult to access and modify. Typically, the data is only available to the device itself for its internal purposes and it must be administered using either a proprietary interface or a standard protocol against a proprietary schema. This leads to many problems, most notably: the need for data replication and difficult interoperation with other devices and applications. MetaComm addresses these problems by providing a framework to integrate data from multiple devices into a meta-directory. The system allows user information to be modified through a directory using the LDAP protocol as well as directly through two legacy devices: a Definity® PBX and a voice messaging system. In order to prevent data inconsistencies, updates to any system must be reflected appropriately in all systems. This paper describes how MetaComm maintains consistency when data integration is performed across several systems with no triggers and with extremely weak typing and transactional support. We also discuss implementation details and experiences.

1. Introduction

Directory Enabled Networking (DEN) [7] simplifies a wide variety of tasks including provisioning network services, allocating resources, reporting, managing end-to-end security, and offering mobile users customized features [6]. While this technology is not limited to LDAP directories [11, 26] or to any particular standard[1], it is frequently associated with efforts by equipment and software vendors to standardize LDAP schemas to support DEN. To supply all the functionality that users expect, middleware to integrate the LDAP directories with network and telecommunication devices is needed. This integration makes data that has traditionally been buried in network/telecommunication devices like routers, PBXs, and messaging platforms available to new applications that can add value to the data. In addition, since much of this data is replicated in multiple devices, corporate directories, and provisioning systems, integration reduces the need to manually re-enter such data, and consequently it reduces data inconsistencies across repositories.

MetaComm makes it far easier to query and modify common data in the devices. The MetaComm system integrates data from multiple telecom devices into an LDAP directory server, making it possible to manage these devices using the LDAP protocol. As a result, MetaComm allows voice products to be integrated with data products through DEN. Besides maintaining data consistency across multiple devices, MetaComm also makes it far easier to modify common data in the devices than is currently possible using legacy interfaces. It allows users to choose any tool that can perform LDAP updates for handling their updates (*e.g.,* a Web browser).

In this paper we describe our initial effort where user data from two legacy devices, a Definity® PBX and a messaging platform, are integrated into a meta-directory. User data is the most valuable and the most likely to be duplicated, so it was a natural first choice. As shown in Figure 1, our implementation allows user data to be modified in two ways: the data can be modified through an LDAP directory which materializes the data from legacy devices; and users can continue to modify the telecommunication devices directly through existing, often proprietary, interfaces. Offering multiple paths to modify parameters in crucial telecommunication devices preserves the experience base of device administrators. In addition, it increases the overall reliability and availability of the system, since updates can still be

[1]In fact, Novell Directory Service has supplied many of the DEN capabilities for some time.

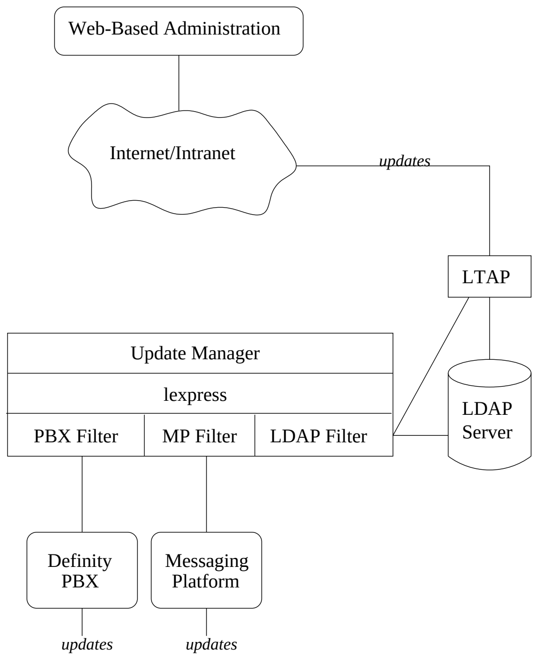

Figure 1. Architecture of MetaComm

made directly to the device even if the directory becomes inaccessible.

However, allowing multiple update paths also adds complexity to the system, especially since some platforms lack triggers and only provide weak typing and transactional support. MetaComm addresses these issues by using: (1) well-known techniques for materialized views and updating through views (*e.g.*, [17]); (2) clever schema design and update ordering; and (3) new tools such as *lexpress* [23], for performing schema translation and integration, and LTAP [19], which provides trigger support for LDAP directories.

The main contributions of this paper include techniques we developed to handle the transactional weaknesses and integration of the underlying systems, as well as the novel combination of existing techniques. The paper is organized as follows. Section 2 gives an overview of the LDAP protocol. Section 3 reviews related data integration work. Section 4 describes the architecture of the MetaComm system. Our experiences in building MetaComm, the trade-offs, and alternatives for a number of the architectural choices are described in Section 5. Section 6 surveys related work. We conclude in Section 7 with some future directions.

2. LDAP overview

LDAP is a widely deployed directory access protocol with implementations by a large number of vendors (see [13] for a partial list). From a database perspective, LDAP can be thought of as very simple query and update protocol. Compared to traditional relational databases, LDAP has some benefits in that it deals well with heterogeneity and allows highly distributed data management while keeping data conceptually unified [4].

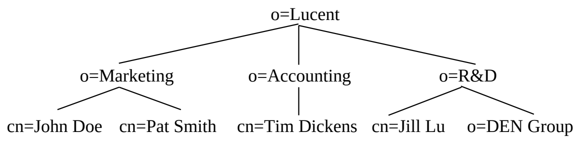

Figure 2. Sample LDAP tree

Directory entries are stored in a tree or forest. LDAP's hierarchical structure makes LDAP directories very scalable — it is straightforward to move an arbitrary sub-tree to its own server.

Figure 2 is an example of a typical tree, simplified to remove all but one attribute from each entry. Each entry in the tree is identified by a *Distinguished Name* (DN) which is a path from the root of the tree to the entry itself. The DN is produced by concatenating the *Relative Distinguished Name* (RDN) of each entry in the path. The RDN for an entry is set at creation time and consists of an attribute name/value pair — or in more complicated cases, a collection of these pairs. The RDN of an entry must be unique among the children of a particular parent entry. For example, in Figure 2, "o=Lucent" and "cn=John Doe" are RDNs, and the DN for John Doe is "cn=John Doe, o=Marketing, o=Lucent". Note the leaf-to-root order is the reverse of that for the representation of a UNIX file or a URL.

The only update commands are to create or delete a single leaf node or to modify a single node. There are two kinds of modification commands: *Modify*, to modify any fields except those appearing in the RDN; and *ModifyRDN*, to modify attribute/value pairs appearing in the RDN. Furthermore, while individual update commands are atomic, one cannot group several update commands into a transaction. For instance, one cannot atomically change a person's name and telephone number if the name is part of the person's RDN but the telephone number is not.

LDAP servers make extensive use of replication to make directory information highly available. Replication and backups are used to handle system and media failure. More traditional database solutions to handle failure and increase availability have also recently become available with Oracle Internet Directory [22].

3. Data integration

The emerging need to provide organization-wide access to data is creating a demand to interconnect previously isolated systems. As a result, integrating information from multiple heterogeneous data sources has become a central issue in modern information systems. A data integration system provides uniform and transparent access to multiple data sources, making information more readily accessible and allowing users to pose queries without having to interact with a specific source using a particular interface.

Even though integrated systems produce many advantages, difficult problems arise when integrating information from multiple sources, most notably: autonomy and heterogeneity. Autonomous systems are often under separate and independent control, using their own data model and application programming interface (API). Heterogeneity can arise at different levels. For instance, different systems may use different APIs, vocabularies (*i.e.,* different systems may use the same term for different concepts or different terms for the same concept), schemas, etc.

There are many steps involved in integrating data from multiple sources:

- *Schema and language translation*: wrappers have to be created for sources to provide access to the underlying data, and mappings between local and global data models are needed to resolve syntactic heterogeneity. The wrappers provide a canonical API and representation for the data in all sources.

- *Schema integration*: schemas corresponding to each source are combined into a single global schema (*e.g.,* [21]). This step resolves structural and semantic heterogeneity (*i.e.,* differences in naming, structure, format, missing/conflicting data, and data interpretation). For example, an example of structural differences occurs when names are represented as a single attribute in one schema and as a composite attribute (*e.g.,* first name and surname) in another. A naming conflict is an example of semantic heterogeneity. It occurs when identical data items are named differently, or semantically different items are named identically.

- *Maintaining consistency and dependencies*: The integration system needs to capture the specification of semantically related data stored in different data sources, so that as updates are applied, these sources remain consistent.

Building custom integration applications that assemble data from appropriate locations is not always a practical solution. It can be prohibitively expensive, inflexible, and hard to maintain. Several research projects have developed *mediator systems* [27] to address these problems (see *e.g.,* [18, 9, 1]). Mediators provide an intermediate layer between the user and the data sources. Each data source is wrapped by software that translates local terms, values and concepts into global concepts shared by some or all sources — smoothing the semantic heterogeneity among the various integrated sources. The mediator then obtains information from one or more wrapped components and exports such information to other components. Queries to the mediator are in a uniform language, independent of the distribution of data over sources and the APIs of the source.

In designing MetaComm, we used several ideas from existing mediator systems for the actual integration. For example, like in the Information Manifold [18], each data source has an associated *description file* that defines the mapping of the local schema into the global schema, as well as constraints on values and mappings of local values into global values. However, there are important differences. Whereas most of the work on mediators concentrates on read-only queries, MetaComm must handle updates. In addition, unlike mediators where queries posed against the unified system are dynamically executed at the various data sources, because of reliability and performance requirements, MetaComm *materializes* subsets of the data from the various sources in an integrated directory. Even though this approach at a first glance resembles data warehouses [12], MetaComm must do much more than a data warehouse. Besides propagating updates from the data sources to the materialized directory, MetaComm must also propagate updates that are applied to the meta-directory to the various data sources.

In the next section, we give a detailed description of the architecture of MetaComm and discuss how the various issues of data integration are addressed in the system.

4. Architecture

MetaComm is a data integration system that creates an integrated materialized view of data from independent, heterogeneous repositories. The main challenge of MetaComm is to foster the cooperation of the multiple repositories, ensuring that data is kept consistent when updates are applied to the various repositories, including the materialized view of the integrated schema.

Figure 1 shows the various components of MetaComm. The integrated schema of MetaComm is an extension of a standard X.500 class [3] that describes people, with auxiliary classes to represent device specific information (details about the schema are given in Section 5.2). The materialized view of the integrated information is stored in an LDAP server.

The Update Manager (UM) is the central component of the system — it ensures that the data in the devices and in the LDAP server are consistent. Consistency is not just a matter of applying the same update to each data repository in a global transaction. Because the repositories lack most basic transaction facilities, MetaComm cannot support traditional transaction semantics. Instead, it uses other techniques to ensure that the repositories converge to the same values after some delay [25, 5]. For example, in MetaComm updates may be applied more than once on certain repositories to ensure correct update ordering, and resynchronization of repositories is used for recovery from catastrophic communication or storage errors. (Directory systems, such as LDAP, maintain a relaxed write-write consistency by ensuring that updates eventually result in the same values for object attributes being present in each copy of the object. MetaComm extends this relaxed write-write consistency to meta-directory updates by reapplying updates that are initially applied in different orders at different directories. When directory applications require read-write consistency, they must supply the transaction discipline necessary to ensure that consistency. Our LTAP work provides one approach to enhancing directory transaction capabilities.)

Maintaining the consistency of the repositories also requires that the semantics of the data are properly reflected in each repository. A filter or wrapper is associated with each repository. In MetaComm there are three such filters, the PBX filter, the Messaging Platform (MP) filter and the LDAP filter, depicted in Figure 1. Each filter has a protocol converter for communicating with its associated repository and a mapper for translating update commands to the schema of the repository. The schema translation and integration of the mapper are realized through *lexpress* (described in Section 4.2). lexpress uses semantic characteristics of the data to provide better data integration. In particular, lexpress uses *data dependencies* to propagate data wherever it is needed in the global or device schema, and *partitioning constraints* to translate schema updates correctly and route them to the proper repositories.

Each repository in the system (*i.e.*, legacy device or LDAP directory server) must notify the UM when a change occurs. The LTAP module adds active functionality to the LDAP server and notifies the UM of changes to data in the LDAP directory (see Section 4.3 for details). The main thread of the UM, the coordinator, responds to update and synchronization requests by propagating update commands to the appropriate filters. The mapper component in the filter further analyzes the request to ensure that updates are properly forwarded to the associated data repository.

Also shown in Figure 1 is the Web-Based Administration (WBA), which provides a single point of administration for the telecom devices. It is worth pointing out that any LDAP tool can contact LTAP to administer the telecom devices, for example, any LDAP enabled Web browser.

4.1. Filters for data sources

In MetaComm, a *filter* is associated with each repository type. Each filter has two components: a protocol converter and mapper. The *protocol converter* provides a unified API for all repositories, which consists of:

- a method to retrieve a record given its key (or id);
- the ability to receive notifications from the device; and
- methods to add, modify and delete records in the device.

Additionally, if a repository is to be synchronized with another repository, in particular a device with the LDAP server, the API must also provide a method to retrieve all relevant data from the repository. (A device is *synchronized* with the LDAP directory when its data is initially loaded into the directory. It is also synchronized with the directory after the directory and the device have temporarily become unable to communicate with each other, and updates that should have been sent from one to the other have been lost — this can occur due to process crash or network problems.)

The second component of filters, the *mapper*, uses the information available in the lexpress description file (*e.g.*, set of attributes, keys, mapping rules) to translate update requests expressed in lexpress' canonical form into updates against the relevant repository. When a filter receives a change notification from its associated repository, it creates a lexpress update descriptor of the change. The UM coordinator chooses the appropriate filters to receive the descriptor. When a filter receives a descriptor, it uses lexpress to translate and apply the update. This separation between protocol and mapping allows protocol-specific software to be reused with varying schema.

4.2. lexpress

MetaComm uses lexpress to describe the mapper component for the various filters. lexpress is a tool for schema translation and integration whose declarative mapping language supports string operations and table translations of attributes, alternate attribute mappings, multi-valued attribute processing, and pattern matching. Matching the pattern of input attributes allows mappings to be resilient when faced with dirty data. Patterns allow mappings to be refined incrementally with a list of special cases.

Users create mappings in the lexpress language that specify the relationship between two schemas as well as other update requirements. Mappings are specified from a source schema to a target schema, so two lexpress mappings are specified for each schema pair. The same filter can be used with multiple schema pairs if the protocols for communicating with the target of the update in each pair are the same. Only the lexpress mapping, which is input data to the lexpress routines, needs to change to accommodate different schema or different versions of the same schema.

lexpress supports propagation of changes to wherever they are needed. Since setting one attribute may affect a set of related attributes, lexpress calculates the transitive closure of the attribute mappings. For example, the LDAP attributes telephoneNumber and DefinityExtension are related through the Definity® attribute Extension. If either changes, lexpress changes the other when the update is propagated to the LDAP Server.

The transitive closure can also propagate changes to other devices in the meta-directory. For example, consider two lexpress mappings: one from the extension for a telephone on a PBX to a telephone number in the LDAP directory, and another from the telephone number to a voice mailbox identifier in the voice messaging platform. When the extension of an existing object changes, the PBX-to-LDAP lexpress mapping requires lexpress to change the telephone number. Because lexpress processes the transitive closure of mappings, it also uses the LDAP-to-MP mapping to change the voice mailbox identifier.

Integration conflicts arise when a client explicitly updates multiple attributes in a transitive closure inconsistently. When such a conflict arises, the first mapping in the transitive closure to be satisfied sets all other unset attributes in the transitive closure. The algorithm does not change the values of explicitly set attributes. In the earlier example, changes to the telephoneNumber or DefinityExtension in the LDAP schema cause the Extension in the Definity to change and vice versa. If telephoneNumber and DefinityExtension are set inconsistently, *i.e.*, they map to different values for Extension, then the first mapping satisfied, *e.g.*, telephoneNumber to Extension, sets Extension. The other mapping in the closure, *e.g.*, Extension to DefinityExtension, is not executed and DefinityExtension retains its new value. Thus, the inconsistently set attributes do not affect each other's values and only one of them has its value propagated to other attributes. We are currently enhancing lexpress to identify cyclic dependencies that do not reach a fixpoint and take appropriate action, at compile time (if a fixpoint can never be reached) or at execution time (if a fixpoint will not be reached for a current update).

Another useful feature of lexpress is the support for partitioning constraints — it automatically migrates data to the right object manager for the data. When an update is sent to a target system, lexpress transforms each update to the correct series of add, delete and modify operations to migrate data to the proper destination. For example, when a person's telephone number changes, the Definity® PBX that manages the person's extension may also change. In this case lexpress translates a modification of a telephone number into two updates: a deletion in one PBX and an add in another PBX.

In general, when a modification of an existing object is requested, lexpress checks the partitioning constraints against both the old and new attributes of the object. For example, when a particular PBX accepts updates for phone numbers beginning with "+1 908-582-9", lexpress checks the old phone number for the object to determine that the object was stored in the PBX and the new attributes for the object to determine that object is still stored in the PBX. Depending on the combination of constraint satisfaction by the old and new attributes, different operations are done on the target directory. Specifically, if the old attributes violate the constraints and the new attributes satisfy them, then the update is forwarded as an add to the target because the object was not previously managed by the target. If the old and new attributes satisfy the constraints, then the update is forwarded as a modify to the target. If the old, but not the new, attributes satisfy the constraints, then the update is forwarded a delete to the target. If neither set of attributes satisfy the constraints, the operation is skipped at the target because the object is not under the target's management.

The components of lexpress are a declarative language for specifying the relationship between two schemas, a compiler that generates machine-independent byte code from the declarative language, and an interpreter for executing the byte codes. The compiler and interpreter are available in a subroutine library that can be called from any program. A library of common mappings for telecommunications directories is available. Descriptions for new sources or changes to descriptions for old sources can be added dynamically (to running programs) by compiling them at run-time using the appropriate lexpress routine. Experience with the language indicates that a few minutes are sufficient to map a new source to the global schema and vice versa. For more details on lexpress the reader is referred to [23].

4.3. Lightweight trigger access process

LDAP servers currently available provide no support for triggers. In MetaComm we used LTAP [19] as a portable solution to add active functionality to LDAP servers. LTAP works as a gateway that *pretends* to be an LDAP server — LDAP commands intended for the LDAP server are intercepted by LTAP which does trigger processing in addition to servicing the original LDAP command. LTAP also provides locking facilities, forbidding updates to an entry while trigger processing is being performed on that entry. In MetaComm, locking is used to help ensure that the devices and directory converge in time to achieve write-write consistency.

4.4. Update manager

The Update Manager (UM) keeps the data in the LDAP directory synchronized with the data in the telecom devices. It responds to update requests that originate from client applications such as the WBA, or from one of the devices, and it ensures that after an update is applied, the information in all devices and directories remains consistent.

As depicted in Figure 1, update requests from client applications such as the WBA are sent to LTAP, which traps the requests and notifies the UM. Update notifications are sent from LTAP to the LDAP filter, which in turn creates a lexpress update descriptor for the update that is then added to a global queue in the UM. The main thread of the UM, the coordinator, iterates through the global update queue, and for each update request, it tells the appropriate filters to generate a sequence of updates to all applicable devices and to the LDAP server. Locking at LTAP (see Section 4.3) blocks conflicting LDAP update requests from being sent to the UM until after the sequence of updates has been applied (*e.g.*, if LTAP receives an update request to an object "cn=John Doe, o=Marketing, o=Lucent", no other LDAP update to this object is allowed to proceed until the UM completes the update sequence and notifies LTAP).

However, no such locks are obtained when updates originate at the devices themselves. A direct device update (DDU) is applied to the device itself. The update is noted during *transaction commit* at the device and a notification is sent to the appropriate device filter. A better alternative would be to have the device alert the UM that an update is being requested and then have the UM queue the request (effectively creating a global ordering for *all* updates). However, this was not practical because the devices must be usable with or without MetaComm. The update sequence for a DDU is as follows:

- the device filter creates a lexpress update descriptor for the update that it forwards to the LDAP filter;
- the LDAP filter translates the descriptor into an update against the LDAP schema and forwards it to LTAP;
- the update is eventually sent back to the UM after proper LTAP locks are obtained.

LTAP is used to obtain locks because the PBX, MP and the LDAP server do not expose their locking capabilities. A consistent ordering of updates is obtained by possibly reapplying the update to the devices. (If updates have occurred at the device entry since the DDU, the update must be reapplied at the device to ensure write-write consistency. The queue maintained by the UM enforces a serialization order.) This technique works because a small number of DDUs are made against any given entry per day. Thus, it is unlikely that a DDU and an overlapping LDAP command will be issued at roughly the same time. If they are, the queue order reapplication quickly resolves the inconsistencies. This technique would not work well if some entries received frequent DDUs. Note that brief inconsistencies between the LDAP server and the device are sometimes created, but quickly eliminated.

If failure occurs while an update is being applied to one of the various devices (*e.g.,* an update is invalid), the update is aborted, an error is logged into the directory, and a notification is sent to the administrator. The administrator can browse through the errors and manually fix the resulting inconsistencies at a later time. A later version of the system will use pre-update information to attempt to undo device updates, making the overall technique akin to sagas [10]. However, logging will always be required for extreme cases, such as when devices and the directory are disconnected for an extended period of time. Note that it is the lack of support for two-phase commit in the underlying repositories that limits the ability of MetaComm to handle these failures.

The UM also supports the synchronization of pre-existing directories. This is necessary to populate the directory initially and to recover from disconnected operations of devices without logging facilities.

4.5. New applications enabled by MetaComm

MetaComm allows modification of PBX/messaging settings through any LDAP tool (there are a variety of GUI interfaces to LDAP directories). For our project, we were able quickly to generate an intuitive Web interface that compares favorably with proprietary interfaces.

Using MetaComm administration, an authorized user/program can easily redirect a telephone extension to a port in another room. An example of using the simplicity of administering telecom devices through MetaComm to produce a hoteling (shared workspaces that are reserved as needed) application is given in [2].

4.6. MetaComm status

MetaComm was included in a demo at InterOp [20]. Lucent has announced a product that will use the MetaComm technology (called Directory Synchronization Technology in the press release) to control Definity® PBXs through an LDAP directory. The technology is currently being transitioned and hardened for commercial use.

5. Experiences

5.1. Maintaining consistency

One of the main issues we faced in designing and developing MetaComm was keeping the various devices consistent with the directory. Since neither LDAP nor the integrated devices provide transaction facilities, all we can assume about these data sources is that an update to a single object is atomic. A number of design decisions were influenced by this deficiency. For example, the integrated LDAP schema had to be designed in such a way to ensure that all attributes that are to be read/written as a unit belong to a single object. Even designing the schema this way did not entirely eliminate non-atomic updates — updates that modify both the RDN and other attributes must be handled by a ModifyRDN/Modify pair of operations. While this is not a problem for updates to the LDAP server (as LDAP cannot be used to express such a pair as a single operation), a DDU may be translated into a pair of LDAP updates. For instance, a direct PBX update might change a person's name (which is used in their RDN) and extension (which is not). Typically, one would expect changes to RDNs to be quite infrequent as attributes like name do not change very often.

Note that locking at the LTAP level prevents the interleaving of operations at the LDAP level. However, if the UM crashes between the ModifyRDN and the Modify operations, the entry will be inconsistent for readers. (Writers will not be able to execute until the UM restarts.) When the UM restarts and re-synchronizes the directory with the devices, the inconsistencies will be eliminated. Note that a UM crash is a catastrophic failure. Furthermore, this problem will only occur in the infrequent case where such a failure occurs at the same time a "complex" DDU update is being applied that modifies both the RDN and some other user data. Such a coincidence of infrequent events is likely to be extremely rare.

In order to provide the synchronization facility (see Section 4.4), MetaComm must guarantee that after a synchronization request is processed, the LDAP server, the device being synchronized, and other devices that share the data being synchronized are consistent. Even though synchronization requests might be viewed as a sequence of individual updates, the set of updates must be applied in isolation, *i.e.,* other updates must not be allowed concurrently. This required two modifications to LTAP. First, LTAP originally only allowed a single update per connection from LTAP to a trigger action server (*e.g.,* UM), but to differentiate synchronization requests from individual updates, persistent connections were added which allow a sequence of updates. Second, in order to guarantee that synchronization requests are executed in isolation, all updates must be disallowed while a synchronization request is being processed. To support this, a new *quiesce* facility was added to LTAP.

5.2. Designing the integrated schema

In designing the integrated schema, we wanted to ensure that it would be easy to add new repositories, and no modifications to standard X.500 classes would be needed. The initial solution we decided upon to meet these criterion was to store all the information related to a person's use of a device (*e.g.,* a PBX) in a child entry of the person in the directory tree. When a new device is added, information about the user/device interaction could be added as a new child. Moreover, most of these user/device entries could use a generic class with lots of optional attributes, rather than creating a new objectclass for each new device. However, the lack of transactions in LDAP forced us to give up this technique. Since many updates to an LDAP directory would require modifying both a parent and a child and these updates cannot be done atomically, we were forced instead to create a new auxiliary objectclass for each new device (to represent user information for that device) and to create new names for the attributes of each auxiliary class.[2]

One practical limitation of auxiliary classes is that they cannot have mandatory attributes. The inability to specify mandatory attributes for auxiliary classes makes it impossible to prevent certain anomolies — like entries whose list of objectclass values indicate that a person uses a PBX, but where no PBX Extension field exists. This will not occur for those who use our tools exclusively. However, users can create such peculiarities easily using off-the-shelf LDAP browsers. Hence, the presence of an auxiliary objectclass only indicates that a person may use a device, not that the person certainly does. To determine more, we must look to see if the PBX Extension field is set, for example. This solution was less elegant than we would have liked, but it does meet the criterion above. It is a general solution for dealing with these kinds of relationships in systems that allow updates through LDAP. If LDAP were extended with transactions, the original solution would be viable as well.

5.3. Limitations of LDAP

As mentioned previously, LDAP has a variety of weaknesses that limits its uses. In addition to the lack of support for triggers (for which LTAP provides a portable solution), LDAP has very weak typing and no transaction support beyond atomic update to a single object. LDAP's chief advantages include scalability and increased flexibility [15], so its disadvantages are closely related to its advantages since full transactions would harm scalability as two-phase commit would be required. However, transactions that allow several entries at a single site to be modified atomically would be a good compromise — solving our atomicity problems while retaining scalability although at the cost of asymmetry. Improving typing with intra-entry constraints would not harm scalability or flexibility and would do much to maintain data quality.

LDAP provides set-valued attributes which could be quite useful in data modeling had they been implemented differently. However, LDAP only allows sets of atomic valued items (*e.g.,* strings). Thus, they are not very useful in practice because there is no way to correlate related fields, *e.g.,* phone numbers and addresses. This inability to correlate fields forced us to forgo the use of set-valued attributes. Instead, we require that a given person have a different directory entry for each location associated with that person. Extending LDAP to allow fields to be arrays or sets of records would solve this problem.

[2]An auxiliary class can be added to an existing object at any time to add new attributes to the object. However, to identify which fields belong to the auxiliary class, unique names for its fields are required.

5.4. Extensions needed to lexpress

In MetaComm, we achieve write-write consistency by reapplying updates to a device that originates the update. For example, if the PBX is updated, it notifies the UM of the update and the UM reapplies the update to the PBX. Problems arose because reapplying add or delete requests to devices where those operations had already occurred produces errors.

lexpress was extended to identify updates that had previously been seen by the device. First, the LDAP schema was extended with a LastUpdater attribute. This attribute is set to the name of the source of an update by the lexpress mappings from a device to the LDAP directory. Each mapping from LDAP to a device was enhanced with a mapping characteristic called Originator that designates which attribute contains the name of the source of the update. When lexpress processes the updates, it returns conditional update operations if updates are being sent to a target that is the same as the source listed in the Originator. Conditional update operations indicate that an update is being repeated and which operation should be used to reapply the update. If a filter knows that the operation is being reapplied at its target, it can take different steps to recover from errors then it would with a normal (*i.e.,* not a reapplied) update. For example, add operations are reapplied as conditional modify operations. If a conditional modify fails, the update filters then attempt to add the record. If a normal modify fails, no add is attempted.

Although the lexpress mappings are simple to construct, we found them to be repetitive for integrating several devices with closely related mappings. A graphical user interface (GUI) was implemented that eliminates the need to enter redundant information. Although transitive closure of dependencies between the source-target pair is automatic in lexpress, transitive closure across all repositories is a matter of design. In particular, all dependencies in a transitive closure must be known in all relevant source-target mappings. We plan to automate the repetition of dependency information in relevant mappings as part of the generation of lexpress description files by the GUI.

5.5. Other issues

Device-generated information Some devices may generate information when an update is applied. For example, when a new extension is added to the messaging platform, a unique id is created which might be needed in other devices. In such situations, the update augmented with the newly generated information might have to be reapplied to other devices — and this process must be repeated until a fixpoint is reached. In MetaComm these cases were simple, because all generated information is only destined for the LDAP server and not for the other device(s). We use lexpress features for communicating changes to the original update and then update the LDAP Server after all other devices are updated.

Running LTAP as a gateway LTAP can be run either as a gateway or as a library that is bound into an application. MetaComm used the gateway approach. We could have coupled MetaComm and LTAP more closely by using the library version. While this would have reduced communication costs between LTAP and the UM, it would have had two disadvantages. First, it would have forced the combined LTAP/UM to process read requests. As it is now, they can run on separate machines and the UM machine does not need to do any read processing. Since LDAP workloads are heavily read-oriented, this offers substantial scalability advantages. Second, upgrades to LTAP would need to be coordinated with the UM. Currently, either LTAP or the UM can be upgraded at any time without affecting the other. This simplifies system upgrades.

6. Related work

Like data warehouses [12], MetaComm *materializes* subsets of the data from the various sources in an integrated directory. One important difference, however, is that the materialized data is also updated and MetaComm must propagate the updates to the various data sources.

There is a large body of research on data integration and a number of prototypes have been built [9, 18, 24] that focus on different aspects of data integration, from semi-automatic wrapper generation to query optimization. However, most of this work focuses on *read-only* queries. MetaComm on the other hand must deal with updates. Nonetheless, many of the ideas from mediator systems have been enhanced to address updates in MetaComm. For example, MetaComm has rules to decide which sources are relevant for a given update; query rewriting is used to translate updates to appropriate formats; and an update execution plan is generated, determining in which order the updates to the various data sources should be applied. Section 3 dicusses the similarities and differences between MetaComm and other data integration systems in more detail.

Schematic heterogeneity has been extensively studied and is well documented in the heterogeneous database research literature [8]. The schema integration component of MetaComm uses the lexpress language to define declarative mappings among disparate schemas from repositories that may have limited querying capabilities.

More details about the LTAP system can be found in [19]. Specifics of how LTAP is used in MetaComm (and changes made to LTAP as a result of our experiences using it in MetaComm) can be found in [2].

MetaComm updates have to be applied to multiple data repositories in a way akin to sagas [10]. Our approach differs from previous work on transactions over heterogeneous sources in that the we have to deal with very weak assumptions, since some sources integrated in MetaComm only support atomic updates to single objects.

Integrated Union Types [16] are used to reconcile data from multiple overlapping sites using virtual views. MetaComm also reconciles data across sources; however, it uses materialized views.

Finally, we should note the growing commercial interest in integration of information, which can be evidenced by products such as Isocor [14] and ZoomIt [28]. The main differences between MetaComm and these products are that MetaComm handles real-time updates and declarative specification of mappers, whereas they support only batch updates and procedural specifications.

7. Conclusions

Even though data integration is a well-studied problem, and there are commercial products that promise to simplify the integration process — it is unlikely that a one-size-fits-all solution to the problem will ever be possible. In this paper, we describe our experiences in integrating data from legacy telecom devices and MetaComm, the system we built to achieve this integration. MetaComm is a full-fledged and extensible mediator system. Its architecture has a modular design, and its various components can be used independently and/or added to other systems.

The first prototype of MetaComm integrates data from PBXs and messaging platforms into an LDAP directory server, and guarantees data consistency while allowing updates to the various independent data repositories. New data sources can be easily added. The extensibility of MetaComm is due mostly to its lexpress component, which handles data conversion, schema integration and data interdependencies in a very elegant and declarative manner.

By providing a simpler and unified interface to data stored in telecom devices, MetaComm greatly simplifies access to this data. As a result, new services and applications, such as hoteling and integrated administration, can be provided with little effort.

Preliminary experiments indicate that MetaComm has acceptable performance for our initial configuration. We are currently investigating its scalability by adding new data sources. Also, the current system uses a very simple security mechanism (based on the security model of LTAP). As future work, we would like to investigate more sophisticated security models.

MetaComm was included in a demo at InterOp [20]. Lucent has announced a product that will use the MetaComm technology (called Directory Synchronization Technology in the press release) to control Definity® PBXs through an LDAP directory. The technology is currently being transitioned and hardened for commercial use.

Acknowledgments:

Thanks to Ravi Sethi and Narain Gehani for management support. Thanks also to Bob Klein and Terry Jennings for helping us understand the details of devices that we integrated. Thanks also to the anonymous reviewer.

References

[1] S. Adali, K. Candan, Y. Papakonstantinou, and V.S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proceedings of SIGMOD*, pages 137–148, 1996.

[2] R. Arlein, J. Freire, N. Gehani, D. Lieuwen, and J. Ordille. Making LDAP active with the LTAP gateway: Case study in providing telecom integration and enhanced services. In *Proc. Workshop on Databases in Telecommunication*, September 1999.

[3] D. W. Chadwick. Understanding x.500 - the directory, 1994. http://www.salford.ac.uk/its024/Version.Web/Contents.htm.

[4] S. Cluet, Olga Kapitskaia, and D. Srivastava. Using LDAP directory caches. In *Proceedings of PODS*, 1999.

[5] A. Demers, D. Greene, A. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. ACM Symp. on the Principles of Distr. Computing*, pages 1–12, August 1987.

[6] Directory Enabled Networking Ad Hoc Working Group. http://murchiso.com/den/.

[7] Customer requirements gathered at the DEN workshop, November 1997. http://murchiso.com/den/general/customer-requirements.html.

[8] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, editors. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann, 1998.

[9] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[10] H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of SIGMOD*, 1987.

[11] T. Howes and M. Smith. *LDAP: Programming Directory-enabled Applications with Lightweight Directory Access Protocol*. Macmillan Technical Publishing, 1997.

[12] W. Inmon. *Building the Data Warehouse*. John Wiley and Sons, 1992.

[13] Innosoft. Innosoft's LDAP world implementation survey. http://www.critical-angle.com/dir/lisurvey.html.

[14] ISOCOR. Metaconnect: Meta-directory solutions for large corporations and service providers. http://www.meta-connect.com.

[15] H. Jagadish, L. Lakshmanan, T. Milo, D. Srivastava, and D. Vista. Querying network directories. In *Proceedings of SIGMOD*, 1999.

[16] V. Josifovski and T. Risch. Integrating heterogeneous overlapping databases through object-oriented transformations. In *Proceedings of VLDB*, pages 435–446, 1999.

[17] A. Keller. *Updating Relational Databases through Views*. PhD thesis, Stanford University, 1995.

[18] A. Y. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of VLDB*, pages 251–262, 1996.

[19] D. Lieuwen, R. Arlein, and N. Gehani. The LTAP trigger gateway for LDAP directories. *Software—Practice and Experience*. To appear.

[20] Lucent Press Release. http://www.lucent.com/press/0599/990503.nsc.html.

[21] R. Miller. Using Schematically Heterogeneous Structures. In *Proceedings of SIGMOD*, 1998.

[22] Oracle Internet Directory. http://www.oracle.com/database/oid.

[23] J. Ordille. Mapping updates for heterogeneous data repositories. Technical report, Bell Laboratories - Lucent Technologies, 1999.

[24] M.T. Roth, M. Arya, L.M. Haas, M.J. Carey, W.F. Cody, R. Fagin, P.M. Schwarz, J. Thomas II, and E.L. Wimmers. The Garlic Project. In *Proceedings of SIGMOD*, page 557, 1996.

[25] L. Seligman and L. Kerschberg. A mediator for approximate consistency: Supporting "good enough" materialized views. *Journal of Intelligent Information Systems*, 8:203–225, 1997.

[26] M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3), December 1997. http://www3.innosoft.com/ldapworld/rfc2251.txt.

[27] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, pages 38–49, March 1992.

[28] Zoomit meta-directory. http://www.zoomit.com.